IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:	§	Group Art Unit: 2186
	§	
Mark Moir	§	Examiner: Tsai, Sheng Jen
Victor M. Luchangco	§	
Maurice Herlihy	§	Atty. Dkt. No.: 6000-33800
	§	
	§	
Serial No.: 10/620,748	§	
	§	
	§	
Filed: July 16, 2003	§	
	§	
For: Obstruction-Free	§	
Synchronization for Shared	§	
Data Structures	§	
	§	

APPEAL BRIEF

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir/Madam:

Further to the Notice of Appeal filed December 16, 2008, Appellants present this Appeal Brief. Appellants respectfully request that the Board of Patent Appeals and Interferences consider this appeal.

I. REAL PARTY IN INTEREST

As evidenced by the assignment recorded at Reel/Frame 014299/0210, the subject application is owned by Sun Microsystems, Inc., a corporation organized and existing under and by virtue of the laws of the State of Delaware, and now having its principal place of business at 4150 Network Circle, Santa Clara, CA 95054.

II. RELATED APPEALS AND INTERFERENCES

No other appeals, interferences or judicial proceedings are known which would be related to, directly affect or be directly affected by or have a bearing on the Board's decision in this appeal.

III. STATUS OF CLAIMS

Claims 1-44 are pending in the application and stand finally rejected. The rejection of claims 1-44 is being appealed. A copy of claims 1-44 is included in the Claims Appendix herein below.

IV. STATUS OF AMENDMENTS

No amendments have been submitted subsequent to the final rejection.

V. SUMMARY OF CLAIMED SUBJECT MATTER

Independent claim 1 is directed to a computer readable storage medium encoding program code executable on one or more processors to implement instantiating a data structure implementation in a memory comprising a double-ended array. (*See, e.g.*, p.1, paragraph [1002], Field of the Invention; FIG. 1, array 110; p. 6, paragraph [1024], make_deque; p.8, paragraph [1030]; and paragraph [1055.1].)

The program code is further executable to implement executing a plurality of opposing-end access operations. The opposing-end access operations, when executed on the one or more processors, access the memory, and provide concurrent push-type and pop-type access to at least one of the opposing ends and concurrent, opposing-end accesses that are non-interfering for at least some states of the array. (*See, e.g.,* p. 4, paragraphs [1014-1015]; pp. 6-7, paragraph [1024], push_right, push_left, pop_right, pop_left; p. 10, paragraph [1036]; and pp. 10-11, paragraph [1038].)

The program code is further executable to implement mediating concurrent execution of the access operations using a single-target synchronization primitive. (*See, e.g.,* p. 4, paragraphs [1014-1015]; and p. 5, paragraph [1019].)

The data structure implementation is linearizable and non-blocking. (*See, e.g.,* p. 2 paragraph [1006]; p. 7, paragraph [1025]; p. 17, paragraph [1053]; and pp. 17-18, paragraph [1056].)

The single-target of the single-target synchronization primitive includes a value encoding for an element of the array and a version number encoded integrally therewith. (*See, e.g.,* pp. 9-10, paragraphs [1034-1035]; and p. 12, paragraphs [1041-1042].)

Independent claim 19 is directed to a computer readable storage medium encoding program code executable on one or more processors to implement a deque.

(*See, e.g.*, p.1, paragraph [1002], Field of the Invention; FIG. 1, array 110; p. 6, paragraph [1024], make_deque; p.8, paragraph [1030]; and paragraph [1055.1].)

The deque implementation is a single-target synchronization primitive based, non-blocking, fully functional deque for which concurrent opposing-end access operations do not always interfere. (*See, e.g.,* p. 4, paragraphs [1014-1015]; p. 5, paragraph [1019]; and paragraph [1055.1].)

Shared storage usage of the deque implementation is insensitive to a number of access operations that concurrently access the deque. (*See, e.g.,* p. 8, paragraph [1031].)

Independent claim 30 is directed to a method of managing obstruction-free access to a shared double-ended array. The method includes instantiating the double-ended array in memory. (*See, e.g.*, p.1, paragraph [1002], Field of the Invention; FIG. 1, array 110; p. 6, paragraph [1024], make_deque; and p. 8, paragraph [1030]; and paragraph [1055.1].)

The method also includes operating on state of the array using access operations that detect interference by other executions thereof using a single-target synchronization primitive. (*See, e.g.*, p. 4, paragraphs [1014-1015].)

The method includes, after detection of an interfering execution, retrying an interfered-with access operation. (*See, e.g.*, FIG. 2; pp. 10-11, paragraph [1037]; FIG. 4; and pp. 15-16, paragraph [1050].)

Execution of respective ones of the access operations allow at least (i) concurrent push-type and pop-type access to at least one of the opposing ends and (ii) concurrent, opposing-end accesses that are non-interfering for at least some states of the array. (*See, e.g.,* p. 4, paragraph [1014]; pp. 6-7, paragraph [1024], push_right, push_left, pop_right, pop_left; pp. 9-10, paragraphs [1035-1036]; and pp. 10-11, paragraph [1038].)

The single-target of the single-target synchronization primitive includes a value encoding for an element of the array and a version number encoded integrally therewith. (*See, e.g.,* pp. 9-10, paragraphs [1034-1035]; and p. 12, paragraphs [1041-1042].)

Independent claim 40 is directed to an apparatus that includes one or more processors and one or more data stores addressable by each of the one or more processors. (*See, e.g.*, p. 1, paragraph [1002], Field of Invention].)

The apparatus also includes means for coordinating concurrent non-blocking execution, by one or more of the processors, of at least opposing-end push-type and pop-type access operations on a fully functional deque data structure encoded in the one or more data stores. (*See, e.g.,* p. 4, paragraphs [1014-1015]; pp. 6-7, paragraph [1024], push_right, push_left, pop_right, pop_left; p. 10, paragraph [1036]; and pp. 10-11, paragraph [1038].)

The coordinating employs a compare-and-swap (CAS) synchronization primitive to detect interference of concurrently executed ones of the access operations. (*See, e.g.*, p. 4, paragraphs [1014-1015].)

The coordinating means ensures that, for all but boundary-condition states of the deque, opposing-end accesses are non-interfering. (*See, e.g.*, p. 5, paragraph [1019]; and pp. 6-7, paragraph [1024].)

The target of the CAS synchronization primitive includes a value encoding for an element of the deque and a version number encoded integrally therewith. (*See, e.g.,* pp. 9-10, paragraphs [1034-1035]; and p. 12, paragraphs [1041-1042].)

Dependent claim 41 is directed to the apparatus of claim 40. The coordinating means tolerates non-progress of interfering executions of the access operations. (*See, e.g.*, p. 2, paragraph [1006]; p. 5, paragraph [1020]; and p. 9, paragraph [1035].)

Dependent claim 42 is directed to the apparatus of claim 40. The apparatus also includes means for managing contention between interfering executions of the access operations. (*See, e.g.,* pp. 5-6, paragraph [1021-1023].)

Independent claim 43 is directed to a non-blocking method of operating on a double-ended queue data structure. (*See, e.g.*, p.1, paragraph [1002], Field of the Invention; FIG. 1, array 110; p. 6, paragraph [1024], make_deque; p.8, paragraph [1030]; and paragraph [1055.1].)

The method includes concurrently executing push-type and pop-type access operations to at least one of opposing ends of the double-ended queue. (*See, e.g.,* p. 4, paragraphs [1014-1015]; pp. 6-7, paragraph [1024], push_right, push_left, pop_right, pop_left; p. 10, paragraph [1036]; and pp. 10-11, paragraph [1038].)

The method includes detecting interference with a particular execution of one of the access operations using a single-target synchronization primitive. (*See, e.g.*, p. 4, paragraphs [1014-1015].)

The method includes tolerating, in the implementation of the double-ended queue data structure, a possibility that two or more executions of the access operations interfere with each other and each consequently fail to make progress. (*See, e.g.*, p. 2, paragraph [1006]; p. 5, paragraph [1020]; and p. 9, paragraph [1035].)

The non-blocking property is achieved while ensuring that, for all but boundary-condition states of the deque, opposing-end accesses are non-interfering and without use of a multi-target synchronization primitive. (*See, e.g.*, p. 5, paragraph [1019]; and pp. 6-7, paragraph [1024].)

The single-target of the single-target synchronization primitive includes a value encoding for an element of the double-ended queue and a version number encoded integrally therewith. (*See, e.g.,* pp. 9-10, paragraphs [1034-1035]; and p. 12, paragraphs [1041-1042].)

The summary above describes various examples and embodiments of the claimed subject matter; however, the claims are not necessarily limited to any of these examples and embodiments. The claims should be interpreted based on the wording of the respective claims.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

1. Claims 1-4, 6-10, 12-14, 16-22, 24, 26-33 and 35-44 stand finally rejected under 35 U.S.C. § 102(e) as being anticipated by Martin et al. (U.S. Publication 2001/0047361) (hereinafter "Martin").

2. Claims 5, 25 and 34 stand finally rejected under 35 U.S.C. § 103(a) as being unpatentable over Martin in view of Rowlands (U.S. Publication 2003/0217115) (hereinafter "Rowlands").

3. Claims 11, 15 and 23 stand finally rejected as being unpatentable over Martin in view of Latour (U.S. Publication 2002/0078123) (hereinafter "Latour").

VII. ARGUMENT

First ground of rejection:

The Examiner rejected claims 1-4, 6-10, 12-14, 16-22, 24, 26-33 and 35-44 under 35 U.S.C. § 102(e) as being anticipated by Martin. Appellants traverse the rejection of these claims for at least the following reasons. Different groups of claims are addressed under their respective subheadings.

Claims 1, 4, 7-10, 12, 16-18, 24, 26, 29-30, 33, 35, 40 and 43:

1. The cited art clearly fails to disclose *wherein the single-target of the single-target synchronization primitive includes a value encoding for an element of the array and a version number encoded integrally therewith.*

On pages 6-7 of the Final Action, the Examiner quotes paragraph [0012] of Martin, which describes the operation of a compare-and-swap (CAS) operation, as allegedly teaching this limitation. The Examiner also cites FIG. 1 of Martin, submitting, "figure 1 of Martin shows that elements of the array has a V number, such as V1, V2, V3 and so on." In these remarks, the Examiner seems to be implying that the V numbers illustrated in FIG. 1 may be targets of a CAS operation, and that they represent version numbers. **The Examiner is incorrect.** The Examiner's own citation in paragraph [0012] includes the following, "The operation fetches and examines the contents V of memory at address A." In addition, paragraph [0077] of Martin describes the V numbers of FIG. 1 this way (emphasis added):

> Values are represented as "V1", "V2", etc. In general, the value field of the illustrated structure may include either a literal value or a pointer value. Particular data structures identified by pointer values are, in general, application specific. Literal values may be appropriate for some applications and, in some realizations, more complex node structures may be employed.

In other words, the V numbers of Martin represent value encodings for respective array elements, not version numbers. Appellants' claim requires that ***the single-target of the single-target synchronization primitive includes a value encoding for an element of the array and a version number encoded integrally therewith***.

Other portions of Martin describe that CAS is used to ensure a precise count of pushes and pops from each end (paragraph [0123]) and is used to set the node's next field to the distinguishing value RY (paragraph [0141]). Paragraph [0019] also includes a general reference to the use of a double compare and swap (DCAS) operation. Appellants assert that none of these passages, or anything else in Martin, describes a version number, much less a version number that is integrally encoded with a value for an element of the array, or that this encoding is included in the single-target of the single-target synchronization primitive used in mediating concurrent execution of the access operations, all of which are required by claim 1. Instead, Martin teaches the use of CAS with a counter unrelated to a version of an array element, and a distinguishing (i.e., fixed) value RY to which a node's next field may be set using CAS.

Appellants assert that there is nothing in Martin describing that the V numbers of Martin encode anything other than a value for each element, and there is no mention of a version number in Martin at all. Thus, Martin clearly fails to anticipate the above-referenced limitation of Appellants' claim.

Anticipation requires the presence in a single prior art reference disclosure of each and every limitation of the claimed invention, arranged as in the claim. M.P.E.P 2131; *Lindemann Maschinenfabrik GmbH v. American Hoist & Derrick Co.*, 221 USPQ 481, 485 (Fed. Cir. 1984). The **identical** invention must be shown in as complete detail as is contained in the claims. *Richardson v. Suzuki Motor Co.*, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989). As discussed above, Martin fails to disclose that *the single-target of the single-target synchronization primitive includes a value encoding for an element of the array and a version number encoded integrally therewith*. Therefore, Martin cannot be said to anticipate claim 1.

For at least the reasons above, the rejection of claim 1 is unsupported by the cited art and removal thereof is respectfully requested. Independent claims 30, 40 and 43 include limitations similar to claim 1, and were rejected for the same reasons as claim 1. Therefore, the arguments presented above apply with equal force to these claims, as well.

Claims 2, 21, and 31:

1. The cited art clearly fails to disclose *wherein the concurrent opposing-end access operations are non-interfering for all but boundary condition states of the array*, as recited in claim 2.

In the Final Action (p. 7), the Examiner cites paragraph [0104] as allegedly teaching this limitation. The Examiner quotes the following passage in paragraph [0104]:

> We begin by describing the operation of "normal" push and pop operations that do not encounter any boundary cases or concurrent operations. Later, we describe special cases for these operations, interaction with concurrent operations, and operations for growing and shrinking the list (emphasis added).

As indicated by the underlined portion above, this passage clearly does not describe concurrent opposing-end access operations, much less that such operations are non-interfering for all but boundary condition states, as required by claim 2. Appellants assert that nothing in Martin teaches the limitations of claim 2.

For at least the reasons above, the rejection of claim 2 is unsupported by the cited art and removal thereof is respectfully requested. Claims 21, and 31 include limitations similar to claim 2, and were rejected for the same reasons as claim 2. Therefore, the arguments presented above apply with equal force to these claims, as well.

Claims 3, 20, and 32:

1. The cited art clearly fails to disclose *wherein the non-blocking implementation is obstruction-free, though not wait-free or lock-free*, as recited in claim 3.

In the Final Action (p. 7), the Examiner cites FIG. 1 as allegedly teaching this limitation, submitting, "as shown in figure 1, the left hat and the right hat are operating on different elements located at the opposite ends, thus obstruction-free; and when both of them try to access the same element, then only one may access and synchronization is needed, hence not wait-free or lock-free." Appellants assert that the Examiner's interpretation of "obstruction-freedom" is clearly inconsistent with the description of this property in Appellants' specification. Obstruction-freedom has nothing to do with whether access operations operate on different elements, or on elements located at the opposite ends of a data structure, as the Examiner implies. Appellants further assert that the Examiner's interpretation that any operations trying to access the same element are necessarily wait-free or lock-free is similarly flawed. As stated in M.P.E.P 2173.01, *"A fundamental principle contained in 35 U.S.C. 112, second paragraph is that applicants are their own lexicographers. They can define in the claims what they regard as their invention essentially in whatever terms they choose so long as any special meaning assigned to a term is clearly set forth in the specification."* In addition, M.P.E.P. 2111 requires that claims be interpreted in light of the specification as it would be interpreted by one of ordinary skill in the art. Appellants assert that nothing in Martin teaches obstruction-freedom, as this property is described in Appellants' specification.

For at least the reasons above, the rejection of claim 3 is unsupported by the cited art and removal thereof is respectfully requested. Claims 20, and 32 include limitations similar to claim 3, and were rejected for the same reasons as claim 3. Therefore, the arguments presented above apply with equal force to these claims, as well.

Claim 6:

1. **The cited art clearly fails to disclose *wherein said mediating comprises attempting to increment the version number included in the single-target of the single-target synchronization primitive*, as recited in claim 6.**

In the Final Action (p. 8), the Examiner cites the CAS and DCAS operations as allegedly teaching this limitation. However, as discussed above in remarks directed to claim 1, there is nothing in Martin that describes the target of these operations including a version number encoded integrally with a value, much less that in mediating, an attempt is made to increment such a version number using a single-target synchronization primitive (e.g., CAS). Appellants further note that DCAS (double compare-and-swap) is clearly not a single-target synchronization primitive, as it operates on two different target locations explicitly identified by operands of the instruction.

For at least the reasons above, the rejection of claim 6 is unsupported by the cited art and removal thereof is respectfully requested.

Claim 13:

1. **The cited art clearly fails to disclose *wherein the opposing-end accesses include opposing-end, push-type accesses; and wherein the boundary-condition states include a nearly full state,* as recited in claim 13.**

In the Final Action (p. 10), the Examiner cites paragraphs [0006] and [0009] as allegedly teaching these limitations. The Examiner quotes the following passage of paragraph [0009]:

> Ideally, operations on one end of the deque would never impede operations on the other end of the deque unless the deque were nearly empty (containing two items or fewer) or, in some implementations, nearly full.

While this passage describes, in general terms, that a deque may be nearly full, it does not describe a boundary-condition state of an array that is indexable as a circular array, as required by claim 13 (as it depends from claim 11).

For at least the reasons above, the rejection of claim 13 is unsupported by the cited art and removal thereof is respectfully requested.

Claim 14:

1. The cited art clearly fails to disclose *wherein distinct left null and right null distinguishing values are employed to identify free elements of the array*, as recited in claim 14.

In the Final Action (p. 10), the Examiner cites figure 1 as allegedly teaching this limitation, submitting, "as shown in figure 1, the left hat and right hat are operating on different elements located at the opposite ends, thus indicating free elements; if both of them are pointing to the same element, then it is an indication that there is no free element." **Appellants assert that these remarks have absolutely nothing to do with claim 14.** Claim 14 recites the use of distinct left null and right null distinguishing values to identify free elements, not merely that the left hat and right hat "are pointing to the same element," as suggested by the Examiner.

For at least the reasons above, the rejection of claim 14 is unsupported by the cited art and removal thereof is respectfully requested.

Claim 19:

1. The cited art clearly fails to disclose *wherein shared storage usage of the deque implementation is insensitive to a number of access operations that concurrently access the deque*, as recited in claim 19.

On p. 11 of the Final Action, the Examiner rejected claim 19 for the same reasons as claim 1. However, the scope of claim 1 and claim 19 are different. For example, claim 1 does not include a limitation similar to the above-referenced limitation of claim 19. **Since the Examiner failed to address the differences between claims 1 and 19, the Examiner has failed to state a *prima facie* rejection of claim 19.** Therefore, the rejection of claim 19 is improper. In addition, Appellants assert that Martin is silent as to the above-referenced limitation of claim 19.

For at least the reasons above, the rejection of claim 19 is unsupported by the cited art and removal thereof is respectfully requested.

Claim 22:

1. The cited art clearly fails to disclose *wherein state of the deque is encoded using an array*, as recited in claim 22.

In the Final Action (p. 12), the Examiner submits that claim 22 recited essentially the same limitations as claim 6, and rejects claim 22 for the same reasons as claim 6. However, claims 6 and 22 recite limitations directed to completely different subject matter. Claim 6 recites, in its entirety, "The storage medium of claim 1, wherein said mediating comprises attempting to increment the version number included in the single-target of the single-target synchronization primitive."

Since the Examiner failed to address the limitations recited in claim 22, the rejection of claim 22 is improper and removal thereof is respectfully requested.

Claims 27 and 41:

1. The cited art clearly fails to disclose *wherein the non-blocking deque implementation does not guarantee that at least one of the interfering concurrently executed access operations makes progress*, as recited in claim 27.

In the Final Action (pp. 12-13), the Examiner cites FIG. 1 as allegedly teaching this limitation, submitting, "as shown in figure 1, the left hat and the right hat are operating on different elements located at the opposite ends, thus obstruction-free; and when both of them try to access the same element, then synchronization is needed, hence not wait-free or lock-free." As discussed in remarks directed to claim 3, this clearly does not teach the obstruction-freedom of Appellants' claims, much less that the deque implementation of Martin does not guarantee that at least one of the interfering concurrently executed access operations makes progress.

For at least the reasons above, the rejection of claim 27 is unsupported by the cited art and removal thereof is respectfully requested. Claim 41 includes limitations similar to claim 27, and was rejected for the same reasons as claim 27. Therefore, the arguments presented above apply with equal force to this claim, as well.

Claims 28, 37, 38, 39, 42, and 44:

1. The cited art clearly fails to disclose *wherein a separate contention management facility is employed to ensure progress in a concurrent computation that employs the deque implementation*, as recited in claim 28.

In the Final Action (p. 13), the Examiner cites the title and the use of a DCAS synchronization primitive in Martin as allegedly teaching this limitation. The title of Martin includes the term "Concurrent" but describes nothing about a separate contention management mechanism (i.e. separate from the deque implementation). The description of a DCAS operation in paragraphs [0018-0019] of Martin teaches that the deque itself may be implemented using a synchronization primitive such as a DCAS operation or using another mechanism, not that a DCAS may be used to implement a separate contention management mechanism. The DCAS operation described in Martin clearly cannot be a **separate contention management mechanism employed to ensure**

progress in a concurrent computation that employs the deque implementation, since it is part of the deque implementation itself.

For at least the reasons above, the rejection of claim 28 is unsupported by the cited art and removal thereof is respectfully requested. Claims 37, 38, 39, 42, and 44 include limitations directed to a separate contention management mechanism, and were rejected for the same reasons as claim 28. Therefore, the arguments presented above apply with equal force to these claims, as well.

Second ground of rejection:

The Examiner rejected claims 5, 25 and 34 under 35 U.S.C. § 103(a) as being unpatentable over Martin in view of Rowlands. Appellants traverse the rejection of these claims for at least the following reasons.

1. The cited art clearly fails to teach or suggest *wherein the single-target synchronization primitive employs a Load-Linked (LL) and Store-Conditional (SC) operation pair*, as recited in claim 5.

In the Final Action (p. 16), the Examiner admits that Martin does not teach this limitation, and relies on Rowlands to allegedly teach it. The Examiner cites paragraphs [0003-0008] of Rowlands, which describe, in general, the use an LL/SC operation pair in atomic read-modify-write operations. Appellants assert that this general reference to LL/SC operation pairs clearly does not suggest the use of an LL/SC operation pair in the specific manner recited in Appellants' claims, i.e., *mediating concurrent execution of the access operations using a single-target synchronization primitive*; and *wherein the single-target of the single-target synchronization primitive includes a value encoding for an element of the array and a version number encoded integrally therewith*. In Rowlands, the LL/SC operation pair is used in a generic read-modify-write operation accessing a shared memory location.

2. The Examiner has not stated a valid reason to combine the references to teach the limitations of claim 5.

The Examiner submits that it would have been obvious for one of ordinary skills in the art at the time of Appellants' invention to use LL/SC as a synchronization mechanism, as demonstrated by Rowlands, and to incorporate it into the existing scheme disclosed by Martin in order to allow other processors to access the memory location for which the atomic read-modify-write is being attempted, "which enhances the overall throughput of the system." Appellants assert that the Examiner's remarks are purely speculative, as no evidence of record describes that the use of LL/SC operation pairs, rather than CAS operations, would enhance the throughput of Martin's system. Appellants assert that the CAS operation described in Martin already allows other processors to access memory locations for read-modify-write operations. Therefore, no other mechanism would be required in the system of Martin.

For at least the reasons above, the rejection of claim 5 is unsupported by the cited art and removal thereof is respectfully requested. Claims 25, and 34 include limitations similar to claim 5, and were rejected for the same reasons as claim 5. Therefore, the arguments presented above apply with equal force to these claims, as well.

Third ground of rejection:

The Examiner rejected claims 11, 15 and 23 under 35 U.S.C. § 103(a) as being unpatentable over Martin in view of Latour. Appellants traverse the rejection of these claims for at least the following reasons.

Claims 11, 15, and 23:

1. The cited art clearly fails to teach or suggest *wherein the array is indexable as a circular array*, as recited in claim 11.

In the Final Action (p. 17), the Examiner admits that Martin does not teach this limitation and relies on Latour as allegedly teaching this limitation. The Examiner cites paragraph [0024], FIG. 9, and paragraph [0054] as teaching the use of a circular linked list in which a sequence of mutexes can be held. Appellants note that Latour teaches the use of locking mechanisms in managing the mutexes so held (see, e.g., Latour's Abstract). Appellants assert that this reference to a circular array of mutexes managed by locks clearly does not teach or suggest the use of a circular array in the double-ended array implementation of Appellants' claims, in which the array implementation is non-blocking, according to the limitations of claim 1.

2. The Examiner has not stated a valid reason to combine the references to teach the limitations of claim 11.

The Examiner submits that Latour teaches the motivation of using a circular linked list is "because it provides more flexibility that other types of storage," specifically noting that the number of storage locations in the ring can be readily changed. The Examiner submits, therefore, that it would have been obvious for one of ordinary skills in the art at the time of Appellants' invention to use a circular linked list, and to incorporate it into the existing system of Martin "in order to provide more flexibility." Appellants assert, however, that there is nothing in the evidence of record that teaches or suggests that the system of Martin would benefit from flexibility in the number of storage locations provided by a circular linked list, or that the deque implementation disclosed in Martin does not include similar flexibility as to the number of storage locations. **In fact, it appears that storage locations can be added to the deque implementation of Martin using the "add_right" operation (see, e.g., paragraphs [0116-0121]) .** Therefore,

there would be no reason to modify Martin to use the circular linked list of Latour, as suggested by the Examiner.

For at least the reasons above, the rejection of claim 11 is unsupported by the cited art and removal thereof is respectfully requested. Claims 15, and 23 include limitations similar to claim 11, and were rejected for the same reasons as claim 11. Therefore, the arguments presented above apply with equal force to these claims, as well.

CONCLUSION

For the foregoing reasons, it is submitted that the Examiner's rejection of claims 1-44 was erroneous, and reversal of his decision is respectfully requested.

The Commissioner is authorized to charge any fees that may be due to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/6000-33800/RCK. This Appeal Brief is submitted with a return receipt postcard.

Respectfully submitted,

/Robert C. Kowert/
Robert C. Kowert, Reg. #39,255
Attorney for Appellants

Meyertons, Hood, Kivlin, Kowert & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
(512) 853-8850

Date: February 23, 2009

VIII. CLAIMS APPENDIX

The claims on appeal are as follows.

1. A computer readable storage medium encoding program code executable on one or more processors to implement:

instantiating a data structure implementation in a memory comprising a double-ended array;

executing a plurality of opposing-end access operations that, when executed on the one or more processors, access the memory, and provide concurrent push-type and pop-type access to at least one of the opposing ends and concurrent, opposing-end accesses that are non-interfering for at least some states of the array; and

mediating concurrent execution of the access operations using a single-target synchronization primitive;

wherein the data structure implementation is linearizable and non-blocking, and

wherein the single-target of the single-target synchronization primitive includes a value encoding for an element of the array and a version number encoded integrally therewith.

2. The storage medium of claim 1,

wherein the concurrent opposing-end access operations are non-interfering for all but boundary condition states of the array.

3. The storage medium of claim 1,

wherein the non-blocking implementation is obstruction-free, though not wait-free or lock-free.

4. The storage medium of claim 1,

wherein the single-target synchronization primitive employs a Compare-And-Swap (CAS) operation.

5. The storage medium of claim 1,

wherein the single-target synchronization primitive employs a Load-Linked (LL) and Store-Conditional (SC) operation pair.

6. The storage medium of claim 1,

wherein said mediating comprises attempting to increment the version number included in the single-target of the single-target synchronization primitive.

7. The storage medium of claim 1,

wherein the double-ended array implements a deque.

8. The storage medium of claim 1,

wherein the opposing-end access operations are at least consistent with semantics of a FIFO queue.

9. The storage medium of claim 1,

wherein the boundary-condition states include an empty state.

10. The storage medium of claim 1,

wherein the boundary-condition states include a single element state.

11. The storage medium of claim 1,

wherein the array is indexable as a circular array.

12. The storage medium of claim 11,

wherein the boundary-condition states include a full state.

13. The storage medium of claim 11,

wherein the opposing-end accesses include opposing-end, push-type accesses; and

wherein the boundary-condition states include a nearly full state.

14. The storage medium of claim 1,

wherein distinct left null and right null distinguishing values are employed to identify free elements of the array.

15. The storage medium of claim 14,

wherein the array is indexed as a circular array; and

wherein an additional distinguishing value is employed to facilitate consumption of free elements by push-type operations at either end of the array.

16. The storage medium of claim 1,

embodied as a software component combinable with program code to provide the program code with non-blocking access to a concurrent shared object.

17. The storage medium of claim 1,

wherein the program code is further executable to provide non-blocking access to a concurrent shared object.

18. The storage medium of claim 1,

comprising at least one medium selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium.

19. A computer readable storage medium encoding program code executable on one or more processors to implement:

a single-target synchronization primitive based, non-blocking, fully functional deque implementation for which concurrent opposing-end access operations do not always interfere,

wherein shared storage usage of the deque implementation is insensitive to a number of access operations that concurrently access the deque.

20. The storage medium of claim 19,

wherein the deque implementation is obstruction-free, though not wait-free or lock-free.

21. The storage medium of claim 19,

wherein the concurrent opposing-end access operations are non-interfering for all but boundary condition states.

22. The storage medium of claim 19,

wherein state of the deque is encoded using an array.

23. The storage medium of claim 22,

wherein the array is a circular array.

24. The storage medium of claim 19,

wherein the single-target synchronization includes use of a Compare-And-Swap (CAS) operation.

25. The storage medium of claim 19,

wherein the single-target synchronization includes use of a Load-Linked (LL) and Store-Conditional (SC) operation pair.

26. The storage medium of claim 19,

wherein at least some concurrently executed access operations interfere with each other; and

wherein the interfering concurrently executed access operations are each retried.

27. The storage medium of claim 26,

wherein the non-blocking deque implementation does not guarantee that at least one of the interfering concurrently executed access operations makes progress.

28. The storage medium of claim 27,

wherein a separate contention management facility is employed to ensure progress in a concurrent computation that employs the deque implementation.

29. The storage medium of claim 19,

wherein the program code is executable to implement:

instantiating the deque in memory; and

executing access operations to operate on state of the deque.

30. A method of managing obstruction-free access to a shared double-ended array, the method comprising:

instantiating the double-ended array in memory; and

operating on state of the array using access operations that detect interference by other executions thereof using a single-target synchronization primitive; and

after detection of an interfering execution, retrying an interfered-with access operation,

wherein execution of respective ones of the access operations allow at least (i) concurrent push-type and pop-type access to at least one of the opposing

ends and (ii) concurrent, opposing-end accesses that are non-interfering for at least some states of the array, and

wherein the single-target of the single-target synchronization primitive includes a value encoding for an element of the array and a version number encoded integrally therewith.

31. The method of claim 30,

wherein the concurrent, opposing-end accesses are non-interfering for all but boundary-condition states of the array.

32. The method of claim 30,

wherein execution of the access operations is obstruction-free, though not wait-free or lock-free.

33. The method of claim 30,

wherein the single-target synchronization primitive employs a Compare-And-Swap (CAS) operation.

34. The method of claim 30,

wherein the single-target synchronization primitive employs a Load-Linked (LL) and Store-Conditional (SC) operation pair.

35. The method of claim 30,

wherein the double-ended array includes a representation of a deque; and

wherein the access operations include both push-type and pop-type access operations at both opposing ends of the deque.

36. The method of claim 30, further comprising:

a contention management facility facilitating progress of access operations.

37. The method of claim 36, further comprising:

changing, during the course of a computation involving the shared double-ended array, a contention management strategy employed by the contention management facility.

38. The method of claim 36,

wherein the contention management facility is separable from the single-target synchronization primitive.

39. The method of claim 30,

wherein progress is ensured not by the shared double-ended array, but rather by a separate contention management facility.

40. An apparatus, comprising:

one or more processors;

one or more data stores addressable by each of the one or more processors; and

means for coordinating concurrent non-blocking execution, by one or more of the processors, of at least opposing-end push-type and pop-type access

operations on a fully functional deque data structure encoded in the one or more data stores, the coordinating employing a compare-and-swap (CAS) synchronization primitive to detect interference of concurrently executed ones of the access operations, the coordinating means ensuring that, for all but boundary-condition states of the deque, opposing-end accesses are non-interfering, wherein the target of the CAS synchronization primitive includes a value encoding for an element of the deque and a version number encoded integrally therewith.

41. The apparatus of claim 40,

wherein the coordinating means tolerates non-progress of interfering executions of the access operations.

42. The apparatus of claim 40, further comprising:

means for managing contention between interfering executions of the access operations.

43. A non-blocking method of operating on a double-ended queue data structure, the method comprising:

concurrently executing push-type and pop-type access operations to at least one of opposing ends of the double-ended queue;

detecting interference with a particular execution of one of the access operations using a single-target synchronization primitive; and

tolerating, in the implementation of the double-ended queue data structure, a possibility that two or more executions of the access operations interfere with each other and each consequently fail to make progress,

wherein the non-blocking property is achieved while ensuring that, for all but boundary-condition states of the deque, opposing-end accesses are non-interfering and without use of a multi-target synchronization primitive, and

wherein the single-target of the single-target synchronization primitive includes a value encoding for an element of the double-ended queue and a version number encoded integrally therewith.

44. The method of claim 43, further comprising:

managing the possibility that access operations interfere with each other and consequently fail to make progress using a substitutable contention management facility separable from implementation of the double-ended queue data structure.

IX. EVIDENCE APPENDIX

No evidence submitted under 37 CFR §§ 1.130, 1.131 or 1.132 or otherwise entered by the Examiner is relied upon in this appeal.

X. RELATED PROCEEDINGS APPENDIX

There are no related proceedings.